Real-Time Line Detection Through an Improved Hough Transform Voting Scheme

Leandro A. F. Fernandes, Manuel M. Oliveira

Universidade Federal do Rio Grande do Sul
Instituto de Informática - PPGC - CP 15064
91501-970 - Porto Alegre - RS - BRAZIL
Tel.: +55 (51) 3316-6161
Fax: +55 (51) 3316-7308

Abstract
The Hough transform is a popular tool for line detection due to its robustness to noise and missing data. However, the computational cost associated to its voting scheme has prevented software implementations to achieve real-time performance, except for very small images. Many dedicated hardware designs have been proposed, but such architectures restrict the image sizes they can handle. We present an improved voting scheme for the Hough transform that allows a software implementation to achieve real-time performance even on relatively large images. Our approach operates on clusters of approximately collinear pixels. For each cluster, votes are cast using an oriented elliptical-Gaussian kernel that models the uncertainty associated with the best-fitting line with respect to the corresponding cluster. The proposed approach not only significantly improves the performance of the voting scheme, but also produces a much cleaner voting map and makes the transform more robust to the detection of spurious lines.

Key words: Hough transformation, real-time line detection, pattern recognition, collinear points, image processing

Email addresses: laffernandes@inf.ufrgs.br (Leandro A. F. Fernandes), oliveira@inf.ufrgs.br (Manuel M. Oliveira).


Preprint submitted to Elsevier 5 March 2007
1 Introduction

Automatic detection of lines in images is a classic problem in computer vision and image processing. It is also relevant to computer graphics in applications such as image-based modeling [1,2] and user-interfaces [3]. In vision and image processing, line detection is a fundamental primitive in a wide range of applications including camera calibration [4], autonomous robot navigation [5], industrial inspection [6,7], object recognition [8] and remote sensing [9]. The Hough transform (HT) [10,11] is an efficient tool for detecting straight lines in images, even in the presence of noise and missing data, being a popular choice for the task. By mapping each feature pixel to a set of lines (in a parameter space) potentially passing through that pixel, the problem of identifying line patterns in images can be converted into the simpler problem of identifying peaks in a vote map representing the discretized parameter space. Although conceptually simple and despite the efforts of many researchers in using hierarchical approaches [12,13], affine-transformation-based approaches [14] and FFT architectures [15], real-time performance has only been achieved with the use of custom-designed hardware [16–18]. Moreover, the peak-detection procedure may identify spurious lines that result from vote accumulation from non-collinear pixels. This situation is illustrated in Fig. 1 (c), where redundant lines have been detected, while some well-defined ones have been missed.

This paper presents an efficient voting scheme for line detection using the Hough transform that allows a software implementation of the algorithm to perform in real time on a personal computer. The achieved frame rates are significantly higher than previously known software implementations and comparable or superior to the ones reported by most hardware implementations for images of the same size. The proposed approach is also very robust to the detection of spurious lines. Fig. 1 (d) shows the result obtained by our algorithm applied to the input image shown in Fig. 1 (a). These results are clearly better than the ones obtained with the use of the state-of-the-art Hough transform technique shown in Fig. 1 (c). For instance, note the fine lines on the bottom-left corner and on the upper-left portion of the image in (d). Those results are achieved by avoiding the brute-force approach of one pixel voting for all potential lines. Instead, we identify clusters of approximately collinear pixel segments and, for each cluster, we use an oriented elliptical-Gaussian kernel to cast votes for only a few lines in parameter space. Each Gaussian kernel models the uncertainty associated with the best-fitting line for its corresponding cluster. The kernel dimensions (footprint) increase and its height decreases as the pixels get more dispersed around the best-fitting line, distributing only a few votes in many cells of the parameter space. For a well aligned group (cluster) of pixels, the votes are concentrated in a small region of the voting map. Fig. 2 (b) and (d) show the parameter spaces associated with the set of pixels shown in Fig. 2 (a). Since the influence of the kernels is restricted
Fig. 1. Line detection using the Hough transform (HT). (a) Input image with 512 × 512 pixels. Note the folding checkerboard (see the dark folding line on the left of the clear king). Since the checkerboard halves are not properly aligned (level), its rows define pairs of slightly misaligned lines that cross at the folding line. (b) Groups of pixels used for line detection, identified using a Canny edge detector [19]. Note that the folding line produces two parallel segments. (c) Result produced by the gradient-based HT at 9.8 fps. Note the presence of several concurrent lines (not only two) along the rows, while some important ones are missing. (d) Result produced by the proposed approach at 52.63 fps. Note the fine lines detected at the bottom-left corner and at the upper-left portion of the image. Images (c) and (d) show the 25 most-relevant detected lines.

to smaller portions of the parameter space, the detection of spurious lines is avoided.

The central contribution of this paper is an efficient voting procedure for detection of lines in images using the Hough transform. This approach allows a software implementation to perform in real time, while being robust to the detection of spurious lines.

The remaining of the paper is organized as follows: Section 2 discusses some related work. Section 3 describes the proposed approach, whose results, advantages and limitations are discussed in Section 4. Section 5 summarizes the paper and points some directions for future exploration.
Fig. 2. The conventional versus the proposed voting procedure. (a) A simple image containing approximately straight line segments. (b) A 3D visualization of the voting map produced by the conventional voting scheme. (c) Clustering of pixels into segments. (d) A 3D visualization of the voting map produced by the proposed technique, which is much cleaner than the one shown in (b). The letters indicate the segments that voted for each of the peaks. To improve the visualization, the height of the vertical axis was truncated to 60 votes. The peaks A-B and C-D received 638 and 380 votes, respectively.

2 Related Work

Hough [10] exploited the point-line duality to identify the supporting lines of sets of collinear pixels in images. In his approach, pixels are mapped to lines in a discretized 2D parameter space using a slope-intercept parameterization. Each cell of the parameter space accumulates the number of lines rasterized over it. At the end, the cells with the largest accumulated numbers (votes) represent the lines that best fit the set on input pixels. Unfortunately, the use of slope-intercept requires a very large parameter space and cannot represent vertical lines. Chung et al. [14] used an affine transformation to improve memory utilization and accelerate Hough’s algorithm. However, the results are not real-time and, like the original one, the algorithm cannot handle (almost) vertical lines.
Algorithm 1 Conventional voting process of the Hough transform

Require: $I \{\text{Binary image}\}$
Require: $\delta \{\text{Discretization step for the parameter space}\}$

1: $Votes \leftarrow 0 \{\text{Initialization of the voting matrix}\}$
2: for each feature pixel $I(x, y)$ do
3:   for $0^\circ \leq \theta < 180^\circ$, using a $\delta$ discretization step do
4:     $\rho \leftarrow x \cos(\theta) + y \sin(\theta)$
5:     $Votes(\rho, \theta) \leftarrow Votes(\rho, \theta) + 1$
6:   end for
7: end for

Duda and Hart [11] replaced the slope-intercept with an angle-radius parameterization based on the normal equation of the line (Eq. 1), naturally handling vertical lines and significantly reducing the memory requirements of the algorithm.

$$x \cos(\theta) + y \sin(\theta) = \rho \tag{1}$$

Here, $x$ and $y$ are the coordinates of the pixel in the image, $\rho$ is the distance from the origin of the image’s coordinate system to the line, and $\theta$ is the angle between the image’s $x$-axis and the normal to the line. By restricting $\theta$ to $[0^\circ, 180^\circ)$ and $\rho$ to $[-R, R]$, where $R = \sqrt{w^2 + h^2}/2$, and $w$ and $h$ are the width and height of the image, respectively, all possible lines in the image have a unique representation in the parameter space. In this case, each pixel is mapped to a sinusoidal line. Assuming that the origin of the image coordinate system is at the center of the image, with the $x$-axis growing to the right and the $y$-axis growing down (Fig. 4, left), Algorithm 1 summarizes the voting process for Duda’s and Hart’s Hough transform. Fig. 2 (b) shows a 3D visualization of the voting map for the simple example shown in Fig. 2 (a). The two peaks represent the two main lines approximated by larger pixel segments in the image. One should notice that the discretization of the image and of the parameter space cause the sinusoidal lines to ‘intersect’ at many cells, creating secondary peaks. This effect is further enhanced by the fact that the pixels in each segment are not exactly collinear. As a result, the peak detection step of the algorithm may retrieve some spurious lines, as shown in Fig. 1 (c). In the proposed approach, the per-pixel brute-force voting scheme described in Algorithm 1 is replaced by a per-segment voting strategy, where the votes for each segment are concentrated on an elliptical region (in parameter space) defined by the quality of a line fitting to the set of pixels in the segment. This situation is illustrated in Fig. 2 (c) and (d). Notice that by restricting the votes to smaller regions of the parameter space, we avoid the detection of spurious lines.

Ballard [20] generalized the Hough transform to detect arbitrary shapes and used the orientation of the edges into account to improve accuracy and reduce
false positives. For straight lines, the use of the normal parameterization with
the gradient information (orientation) to limit the range of $\theta$ over which the
associated $\rho$ values should be calculated [21] has become the universally used
version of the Hough transform. Although in spirit this approach is the closest
to ours, it still quite different and casts too many votes.

Many researchers have proposed variations and extensions to the HT. Illing-
worth and Kittler [22] and Leavers [23] provide in-depth discussions of many
of these algorithms. All these variations, however, remained computationally
intensive and unable to execute in real time on current general-purpose proces-
sors. This has inspired the design of dozens of specialized architectures for
HT [17,18], as well as the use, for this purpose, of graphics hardware [8], spe-
cialized image-processing hardware [24], and FFT architectures [15]. Albanesi
et al. [17] presents a comprehensive survey of specialized architectures for
real-time Hough transform. As they point out, ranking these architectures is
a difficult task because the various proposals use different kinds of paramete-
risations, different quantizations of the parameter space, and even different
image sizes. Out of thirty four analyzed architectures, only three were capable
of handling images with dimensions of $1024 \times 1024$ or bigger [25,16] and only
a few can handle $512 \times 512$ images.

The proposed optimization for the Hough transform voting scheme allows a
software implementation running on a PC to exhibit real-time performance,
comparable or superior to the ones obtained by most specialized hardware for
HT.

3 The Kernel-Based Voting Algorithm

As mentioned in the previous section, the biggest source of inefficiency and
false positives in the conventional Hough transform is its brute force voting
scheme. Assuming the existence of an ideal transformation that maps pixels
to a continuous parameter space, one would notice, in the voting map, a set
of main peaks surrounded by smaller ones. The main peaks would correspond
to lines that better represent the image lines, whereas the secondary peaks
would result from the uncertainty due to the discretization of the image space
and to the fact that the feature pixels might not be exactly collinear. This
suggests a simple and natural way of casting votes: (i) starting from a binary
image, cluster approximately collinear feature pixels, (ii) for each cluster, find
its best fitting line and model the uncertainty of the fitting, and (iii) vote
for the main lines using elliptical Gaussian kernels computed from the lines
associated uncertainties. A flowchart illustrating the proposed line detection
pipeline is shown in Fig. 3. Next, we present a detailed explanation of each
stage of this process.
In order to identify clusters of approximately collinear feature pixels in the binary image (which has been obtained using a Canny edge detector [19] plus thresholding and thinning), we link strings of neighbor edge pixels [26] and then subdivide these strings into sets of most perceptually significant straight line segments [27]. We use the linking procedure implemented by the VISTA image processing library [26], which consists of starting with a seed pixel and creating a chain of connected feature pixels. A pseudo code for this algorithm is presented in Appendix 1. The subdivision of the resulting chains is performed using the technique described in [27], which consists of recursively splitting a string at its most distant pixel from the line segment defined by the string end points. A string split is performed whenever the new sub-strings are better fit by the segments defined by their end points than is the parent string itself, given the constraints that each resulting sub-string might have at least so many pixels (an informed parameter). Thus, each group of almost collinear feature pixels is composed by the pixels in the sub-string between the end points of its defining segment. Although both the linking and subdivision techniques used here ([26] and [27], respectively) are well known, to the best of our knowledge, their combination has not been used previously. Clusters of feature pixels are illustrated in Fig. 2 (c), where each letter is associated with one recovered segment. Notice, however, that different segments have different numbers of pixels, as well as have different degrees of dispersion around their corresponding best-fitting lines. The smaller the dispersion, the more concentrated the votes should be on the voting map. This property is modeled using elliptical-Gaussian kernels, where the dimensions and the height of the kernels are computed based on the uncertainties of the corresponding segments. Therefore, the footprint of a kernel associated with a more collinear cluster of pixels will be smaller than for a more disperse one, thus concentrating more votes on a smaller portion of the voting map. The details about how to...
Fig. 4. Coordinate system used to avoid vertical lines during line fitting using linear regression. Original cluster and its eigenvectors $u$ and $v$ (left). The new coordinate system is defined by $\bar{p}$, the centroid of the pixels (new origin), and by the rotated eigenvectors $u'$ and $v'$ (right). The best-fitting line is guaranteed to be horizontal.

**3.1 Computing the Elliptical Kernels**

Given a cluster $S$ of approximately collinear pixels in an image, its votes are cast around the cell (in parameter space) that corresponds to the cluster’s best-fit line $l$. $l$ is computed from $\bar{p} = (x_{\bar{p}}, y_{\bar{p}})^T$, the centroid of the pixel distribution, and $u = (x_u, y_u)^T$, its eigenvector with the largest eigenvalue. This situation is illustrated on the left part of Fig. 4. $v = (x_v, y_v)^T$, the second eigenvector, is perpendicular to $l$, allowing one to write:

$$Ax + By + C = x_v x + y_v y - (x_v x_{\bar{p}} + y_v y_{\bar{p}}) = 0$$

Comparing Eq. (1) and (2), one obtains the values for $\rho$ and $\theta$:

$$\rho = -C = x_v x_{\bar{p}} + y_v y_{\bar{p}}$$
$$\theta = \cos^{-1}(A) = \cos^{-1}(x_v)$$

Equation (3)

Since $\theta \in [0^\circ, 180^\circ]$, $\sin(\theta) \geq 0$. Also from Eq. (1) and (2), one has $\sin(\theta) = y_v$. Therefore, we need to enforce that $y_v \geq 0$ so that the coordinate system and the limits of the parameter space match the ones defined by [11].

The footprint size used for casting votes for $S$ around $(\rho, \theta)$ is given by the uncertainty associated to the fitting of $l$ to $S$. This uncertainty is represented by the variances $\sigma^2_\rho$ and $\sigma^2_\theta$, which are used to define the shape and extension of the corresponding elliptical-Gaussian kernel $G$. The orientation of the kernel is given by the covariance $\sigma_{\rho\theta}$. Clusters formed by fairly collinear pixels will present small variances, thus concentrating their votes in small regions of the parameter space, such as clusters $A$, $B$, $C$ and $D$ in Fig. 2 (d). Clusters
presenting bigger variances will spread their votes over larger regions of the parameter space, such as clusters \( E, F \) and \( G \).

One way to compute the variances and covariance for \( \rho \) and \( \theta \) is to rewrite Eq. (1) as

\[
y = mx + b = -\frac{\cos(\theta)}{\sin(\theta)} x + \frac{\rho}{\sin(\theta)}
\]

and use linear regression to estimate the coefficients \( m \) and \( b \), as well as their variances \( \sigma_m^2 \) and \( \sigma_b^2 \), and covariance \( \sigma_{mb} \), from which \( \sigma_{\rho}^2 \), \( \sigma_{\theta}^2 \) and \( \sigma_{\rho \theta} \) can be computed. However, Eq. (4) is not defined when \( \sin(\theta) = 0 \). Moreover, the quality of the fitting by linear regression decreases as the lines approach the vertical direction [28]. We avoid these problems by mapping each cluster to a new coordinate system, whose origin is defined by \( \vec{p} \) and whose horizontal axis is given by the rotated eigenvector \( u' \). This transformation is depicted in Fig. 4 and guarantees that the best-fitting line is always horizontal.

Eq. (5) and (6) show how to map pixels and lines, respectively, to the new coordinate system. Note that the coefficients of the line equations need to be transformed using the inverse transpose of the operation applied to the pixels. This situation is analogous to the transformations of vertices and normal vectors performed by the graphics pipeline (the vector \( (A, B) \) is perpendicular to \( l \)).

\[
p_i' = \begin{pmatrix} x_{p_i}' \\ y_{p_i}' \\ 1 \end{pmatrix} = RT p_i 
\]

\[
l' = \begin{pmatrix} A' \\ B' \\ C' \end{pmatrix} = \left( (RT)^{-1} \right)^T l = R \left( T^{-1} \right)^T l
\]

Here, \( p_i \) are the pixel coordinates represented in homogeneous coordinates, and \( l = (A, B, C)^T \) (Eq. 2). \( R \) and \( T \) are the following rotation and translation matrices:

\[
R = \begin{pmatrix} x_u & y_u & 0 \\ x_v & y_v & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad T = \begin{pmatrix} 1 & 0 & -x_{\bar{p}} \\ 0 & 1 & -y_{\bar{p}} \\ 0 & 0 & 1 \end{pmatrix}
\]
Applying linear regression to the set of \( n \) transformed pixels from cluster \( S \), one can estimate \( m' \), \( b' \) and their associated variances and covariance [28]:

\[
\begin{align*}
\sigma^2_{m'} &= \frac{S_{\sigma}}{\Delta}, \sigma^2_{b'} = \frac{S_{x^2}}{\Delta}, \sigma_{m'b'} = \frac{S_{x}}{\Delta}
\end{align*}
\] (8)

where

\[
\begin{align*}
S_{\sigma} &= \sum_{i=0}^{n-1} \frac{1}{\sigma_i^2}, \quad S_{x^2} = \sum_{i=0}^{n-1} \frac{(x_{p'}^i)^2}{\sigma_i^2}, \\
S_x &= \sum_{i=0}^{n-1} \frac{x_{p'}^i}{\sigma_i^2}, \quad \Delta = S_{\sigma} S_{x^2} - S_x^2
\end{align*}
\] (9)

\( \sigma_i \) is the uncertainty associated to the \( v' \) coordinate of the transformed pixel \( p'_i \) (Fig. 4, right). Assuming that this uncertainty is ±1 pixel (due to image discretization), then \( \sigma^2_i = 1 \). Also, since \( \bar{p'} = (0, 0)^T \) (Fig. 4, right), \( S_x = 0 \), and Eq. (8) is simplifies to

\[
\begin{align*}
\sigma^2_{m'} &= \frac{1}{\sum_{i=0}^{n-1} (x_{p'}^i)^2}, \sigma^2_{b'} = \frac{1}{n}, \sigma_{m'b'} = 0
\end{align*}
\] (10)

Note that Eq. (10) only uses the \( x_{p'} \) coordinate of \( p'_i \). Therefore, Eq. (5) only needs to be solved for \( x_{p'} \). By inverting Eq. (6) and expressing \( A' \), \( B' \) and \( C' \) in terms of \( m' \) and \( b' \), one gets:

\[
\begin{align*}
\begin{pmatrix} A \\ B \\ C \end{pmatrix} &= \left( R \left( T^{-1} \right)^T \right)^{-1} l' = T^T R^T \begin{pmatrix} -m' \\ 1 \\ -b' \end{pmatrix} \\
&= \begin{pmatrix} x_v - x_u m' \\ y_v - y_u m' \\ (x_u x_{\bar{p}} + y_u y_{\bar{p}}) m' - x_v x_{\bar{p}} - y_v y_{\bar{p}} - b' \end{pmatrix}
\end{align*}
\] (11)

From Eq. (11), \( \rho \) and \( \theta \) can be expressed in terms of \( m', b', \bar{p}, \) and the eigenvectors \( u \) and \( v \):

\[
\begin{align*}
\rho &= -C' = x_v x_{\bar{p}} + y_v y_{\bar{p}} + b' - (x_u x_{\bar{p}} + y_u y_{\bar{p}}) m' \\
\theta &= \cos^{-1}(A) = \cos^{-1}(x_v - x_u m')
\end{align*}
\] (12)
Finally, the variances and covariance for $\rho$ and $\theta$ can be estimated from the variances and covariance of $m'$ and $b'$ using a first order uncertainty propagation analysis [29]:

$$
\begin{pmatrix}
\sigma^2_{\rho} & \sigma_{\rho\theta} \\
\sigma_{\rho\theta} & \sigma^2_{\theta}
\end{pmatrix}
= \nabla f
\begin{pmatrix}
\sigma^2_{m'} & \sigma_{m'b'} \\
\sigma_{m'b'} & \sigma^2_{b'}
\end{pmatrix}
\nabla f^T
$$

(13)

where $\nabla f$ is the Jacobian of Eq. (12), computed as shown in Eq. (14). Given the Jacobian, one can evaluate Eq. (13) taking into account the fact that both $m'$ and $b'$ are equal to zero (i.e., the transformed line is horizontal and passes through the origin of the new coordinate system, as shown in Fig. 4, right).

$$
\nabla f = \begin{pmatrix}
\partial \rho / \partial m' & \partial \rho / \partial b' \\
\partial \theta / \partial m' & \partial \theta / \partial b'
\end{pmatrix} = \begin{pmatrix}
-x_u x_{\rho} - y_u y_{\rho} & 1 \\
x_u / \sqrt{1 - x_u^2} & 0
\end{pmatrix}
$$

(14)

Angles should be treated consistently throughout the computations. Note that virtually all programming languages represent angles using radians, while the $(\rho, \theta)$ parameterization uses degrees. Thus, one should convert angles to degrees after they are computed using Eq. (3) and take this transformation into consideration when evaluating the partial derivative $\partial \theta / \partial m'$, which is used in the computation of the Jacobian matrix $\nabla f$ (13).

3.2 Voting Using a Gaussian Distribution

Once one has computed the variances and covariance associated with $\rho$ and $\theta$, the votes are cast using a bi-variated Gaussian distribution given by [30]:

$$
G_k (\rho_j, \theta_j) = \frac{1}{2\pi \sigma_\rho \sigma_\theta \sqrt{1 - r^2}} e^{\left(-\frac{z}{2(1-r^2)}\right)}
$$

(15)

where

$$
\begin{align*}
z &= \frac{(\rho_j - \rho)^2}{\sigma^2_\rho} - \frac{2r (\rho_j - \rho)(\theta_j - \theta)}{\sigma_\rho \sigma_\theta} + \frac{(\theta_j - \theta)^2}{\sigma^2_\theta} \\
r &= \frac{\sigma_{\rho\theta}}{\sigma_\rho \sigma_\theta}
\end{align*}
$$

Here, $k$ is the k-th cluster of feature pixels, $\rho_j$ and $\theta_j$ are the position in the parameter space (voting map) for which we would like compute the number
of votes, $\rho$ and $\theta$ are the parameters of the line equation (3) of the best-fitting line for cluster $k$, and $\sigma^2_{\rho}$, $\sigma^2_{\theta}$ and $\sigma_{\rho\theta}$ are the variances and covariance of $\rho$ and $\theta$ (Eq. 13).

In practice, Eq. (15) only needs to be evaluated for a small neighborhood of $\rho_j$ and $\theta_j$ values around $(\rho, \theta)$ in the parameter space. Outside this neighborhood, the value returned by Eq. (15) will be negligible and can be disregarded. The threshold $g_{\text{min}}$ used for this is computed as:

$$g_{\text{min}} = \min \left( G_k \left( \rho + \rho_w \sqrt{\lambda_w}, \theta + \theta_w \sqrt{\lambda_w} \right) \right)$$  \hspace{1cm} (16)$$

where $w = (\rho_w, \theta_w)^T$ is the eigenvector associated to $\lambda_w$, the smaller eigenvalue of the covariance matrix shown on the left side of Eq. (13). $\lambda_w$ represents the variance of the distribution along the $w$ direction. Intuitively, we evaluate Eq. (15), starting at $(\rho, \theta)$ and moving outwards, until the returned values are smaller than $g_{\text{min}}$, the value produced at the border of the ellipse representing the footprint of the kernel computed in Section 3.1. Fig. 5 illustrates the inside-going-out order used to evaluate Eq. (15) during the voting procedure.

The complete process of computing the parameters used for the elliptical-Gaussian kernels is presented in Algorithm 2. Special attention should be given to lines 21 and 22, where the variances $\sigma^2_{\rho}$ and $\sigma^2_{\theta}$, used to compute the kernel dimensions in Algorithm 4, are scaled by $2^2$. Such a scaling enforces the use of two standard deviations (i.e., $2\sigma_{\rho}$ and $2\sigma_{\theta}$, respectively) when computing the kernel’s footprint. This gives a 95.4% assurance that the selected region of the parameter space that receives votes covers the true line represented by the cluster of feature pixels in the image. If the pixels are exactly collinear, such variances will be zero. This can happen in the case of horizontal and vertical lines represented by rows and columns of pixels, respectively. In this case, a variance value of $2^{40.1}$ (i.e., a small variance of 0.1 scaled by $2^2$) is assigned to $\sigma^2_{\theta_k}$ in order to avoid division by zero in Eq. (15).
Algorithm 2 Computation of the Gaussian kernel parameters

Require: $S \{Groups \ of \ pixels\}$

1: for each group of pixels $S_k$ do
2: $p \leftarrow$ pixels in $S_k$ {Pixels are 2D column vectors}
3: $n \leftarrow$ number of pixels in $S_k$
4: {Alternative reference system definition}
5: $\bar{p} \leftarrow$ mean point of $p$ {Also a column vector}
6: $V, \Lambda \leftarrow$ eigen($\left( p - \bar{p} \right) \left( p - \bar{p} \right)^T$) {Eigen-decomposition}
7: $u \leftarrow$ eigenvector in $V$ for the biggest eigenvalue in $\Lambda$
8: $v \leftarrow$ eigenvector in $V$ for the smaller eigenvalue in $\Lambda$
9: {$y_v \geq 0$ condition verification}
10: if $y_v < 0$ then
11: $v \leftarrow -v$
12: end if
13: {Normal equation parameters computation (3)}
14: $\rho_k \leftarrow \langle v, \bar{p} \rangle$
15: $\theta_k \leftarrow \cos^{-1}(x_v)$
16: {$\sigma^2_{m'}$ and $\sigma^2_{b'}$ computation, substituting Eq. (5) in (10)}
17: $\sigma^2_{m'} \leftarrow 1/\sum \langle u, p_i - \bar{p} \rangle^2$
18: $\sigma^2_{b'} \leftarrow 1/n$
19: {Uncertainty from $\sigma^2_{m'}$ and $\sigma^2_{b'}$ to $\sigma^2_{\rho_k}$, $\sigma^2_{\theta_k}$ and $\sigma_{\rho_k\theta_k}$}
20: $M = \nabla f \begin{pmatrix} \sigma^2_{m'} & 0 \\ 0 & \sigma^2_{b'} \end{pmatrix} \nabla f^T$ {\nabla f matrix in Eq. (14)}
21: $\sigma^2_{\rho_k} \leftarrow 2^2 M_{(1,1)}$
22: $\sigma^2_{\theta_k} \leftarrow 2^2 M_{(2,2)}$ if $M_{(2,2)} \neq 0$, or $2^20.1$ if $M_{(2,2)} = 0$
23: $\sigma_{\rho_k\theta_k} \leftarrow M_{(1,2)}$
24: end for

3.3 The Voting Procedure

Once the parameters of the Gaussian kernel have been computed, the voting process is performed according to Algorithms 3 and 4. Here, $\delta$ is the discretization step used for both dimensions of the parameter space, defined so that $\rho \in [-R, R]$ and $\theta \in [0^\circ, 180^\circ - \delta]$. Algorithm 3 performs the voting process by splitting the Gaussian kernel into quadrants, as illustrated in Fig. 5. The ellipse in Fig. 5 represents the kernel footprint (in parameter space), with its center located at the best-fitting line for the cluster of pixels. The arrows indicate the way the voting proceeds inside quadrants I to IV, corresponding to the four calls of the function $Vote()$ in lines 17-20 of Algorithm 3.

During the voting procedure, special attention should be given to the cases where the kernel footprint crosses the limits of the parameter space. In case $\rho_j$ is outside $[-R, R]$, one can safely discard the vote, as it corresponds to a line outside of the limits of the image. However, if $\theta_j \not\in [0^\circ, 180^\circ)$ the voting
Fig. 6. Special care must be taken when the $\theta_j$ parameter is close to 0° or to 180°. In this situation, the voting process for line continues in the diagonally opposing quadrant, at the $-\rho_j$ position.

should proceed in the diagonally opposing quadrant in the parameter space. This special situation happens when the fitted line is approximately vertical, causing $\rho_j$ to assume both positive and negative values. As $\rho_j$ switches signs, the value of $\theta_j$ needs to be supplemented (i.e., $180^\circ - \theta_j$). This situation is illustrated in Fig. 6, where the cluster of pixels shown on the left casts votes using the wrapped-around ellipse on the right. In Algorithm 4, the cases where the kernel crosses the parameter space limits are handled by lines 23 (for $\rho_j$) and 7-19 (for $\theta_j$).

Voting accumulation can be performed using integers. This is achieved scaling the result of Eq. (15) by $g_s$ (lines 22 and 26 of Algorithm 4). $g_s$ is computed in line 15 of Algorithm 3 as the reciprocal of $g_{min}$ (16). Such a scaling guarantees that the each cell in the parameter space covered by the Gaussian kernel will receive an integer number of votes. The use of an integer voting map optimizes the task of peak detection.

Another important optimization in Algorithm 3 is a culling operation that helps to avoid casting votes for kernels whose contributions are negligible. In practice, such kernels do not affect the line detection results, but since they usually occur in large numbers, they tend to waste a lot of processing

Fig. 7. Cumulative histogram for the normalized heights of the kernels associated to the clusters of pixels found in Fig. 1 (a). Using $h_{min} = 0.002$, only 216 (Fig. 8, right) out of the original 765 clusters (Fig. 8, left) pass the culling test. These clusters correspond to the continuous portion of the graph.
Algorithm 3 Proposed voting process. The Vote() procedure is in Algorithm 4.

**Require:** $S \{\text{Kernels parameters computed by Algorithm 2}\}$
**Require:** $h_{min} \{\text{Minimum height for a kernel [0, 1]}\}$
**Require:** $\delta \{\text{Discretization step for the parameter space}\}$

1: \{Discard groups with very short kernels\}
2: $h_{max} \leftarrow$ height of the tallest kernel in $S$
3: $S' \leftarrow$ groups of pixels in $S$ with $\left(\frac{\text{height}}{h_{max}}\right) \geq h_{min}$
4: \{Find the $g_{min}$ threshold. $G_k$ function in Eq. (15)\}
5: $g_{min} \leftarrow 0$
6: for each group of pixels $S'_k$ do
7: \[
M \leftarrow \begin{pmatrix}
\sigma^2_{\rho_k} & \sigma_{\rho_k \theta_k} \\
\sigma_{\rho_k \theta_k} & \sigma^2_{\theta_k}
\end{pmatrix}
\]
8: $V, \Lambda \leftarrow \text{eigen}(M) \{\text{Eigen-decomposition of } M\}$
9: $w \leftarrow$ eigenvector in $V$ for the smaller eigenvalue in $\Lambda$
10: $\lambda_w \leftarrow$ smaller eigenvalue in $\Lambda$
11: $g_{min} \leftarrow \min \left(g_{min}, G_k \left(\rho_k + \rho_w \sqrt{\lambda_w}, \theta_k + \theta_w \sqrt{\lambda_w}\right)\right)$
12: end for
13: \{Vote for each selected kernel\}
14: $\text{Votes} \leftarrow 0 \{\text{Initialization of the voting matrix}\}$
15: $g_s \leftarrow \max(1/g_{min}, 1) \{\text{Scale factor for integer votes}\}$
16: for each $S'_k$ group of pixels do
17: \[
\text{Votes} \leftarrow \text{Vote}(\text{Votes}, \rho_k, \theta_k, \delta, \delta, g_s, S'_k)
\]
18: \[
\text{Votes} \leftarrow \text{Vote}(\text{Votes}, \rho_k, \theta_k - \delta, -\delta, \delta, g_s, S'_k)
\]
19: \[
\text{Votes} \leftarrow \text{Vote}(\text{Votes}, \rho_k - \delta, \theta_k - \delta, -\delta, \delta, g_s, S'_k)
\]
20: \[
\text{Votes} \leftarrow \text{Vote}(\text{Votes}, \rho_k - \delta, \theta_k, -\delta, \delta, g_s, S'_k)
\]
21: end for

time, becoming a bottleneck for the voting and peak detection steps. Such kernels usually come from groups of widely spaced feature pixels. The inherent uncertainty assigned to such groups of pixels results in short kernels (low relevance) with large footprints (responsible for the extra processing). For instance, for the example shown in Fig. 1 (a), the processing time using the culling operation was 9 ms for voting and 10 ms for peaks detection (see Table 2). On the other hand, without culling, these times were 314 ms ($\sim 34 \times$) and 447 ms ($\sim 44 \times$), respectively. These times were measured on a 2.21 GHz PC.

Culling is achieved by taking the height of the tallest kernel and using it to normalize the heights of all Gaussian kernels (defined by Eq. (15)) to the [0, 1] range. Only the kernels whose normalized heights are bigger than a relative threshold $h_{min}$ pass the culling operation (lines 2-3 in Algorithm 3). Thus, the culling process is adaptive as the actual impact of $h_{min}$ varies from image to image, as a function of its tallest kernel. For the examples shown in the paper, we used a threshold $h_{min} = 0.002$, which is very conservative. Still, the
performance impact of the culling procedure is huge. Alternatively, one could perform culling using the following procedure: (i) normalize the kernel heights and sort them in descending order; (ii) compute the cumulative histogram of these heights (Fig. 7); (iii) choose a cutoff point based on the derivative of the cumulative histogram \(\text{e.g.}, \text{when the derivative approaches zero}\); (iv) perform culling by only voting for kernels to the left of the cutoff point. This alternative culling strategy produces similar results to the ones obtained with our approach, but ours has the advantages that there is no need to sort the kernels (the tallest kernel is found in \(O(n)\) time by simply comparing and saving the biggest while the kernels are created), there is no need to compute the histogram, nor to analyze its derivatives.

Fig. 7 shows the cumulative histogram for the normalized heights of the kernels associated to the clusters of pixels found in Fig. 1 (a). Using \(h_{\text{min}} = 0.002\), only 216 (Fig. 8, right) out of the original 765 clusters (Fig. 8, left) pass the culling test. These 216 clusters correspond to the continuous portion of the graph shown in Fig. 7. We have empirically chosen the \(h_{\text{min}}\) value and a conservative choice, like the one used here, is better than a more restrictive parameterization because linear features may be comprised by many small disjoint (even noisy) clusters. However, as shown in Fig. 8 (right), the use of such a conservative estimate allows for non-very-much-collinear clusters to be included in the voting process. On the other hand, the proposed voting algorithm naturally handles these situations, as such clusters tend to produce wide short Gaussian kernels, thus spreading a few votes over many cells. As a result, these kernels do not determine peak formation.

3.4 Peak Detection

To identify peaks of votes in the parameter space, we have developed a sweep-plane approach that creates a sorted list with all detected vote peaks, according to their importance (number of votes).

Given a voting map, first we create a list with all cells that receive at least one vote. Then, this list is sorted in descending order according to the result of the convolution of the cells with a \(3 \times 3\) Gaussian kernel. This filtering operation smooths the voting map, helping to consolidate adjacent peaks as single image lines. Fig. 9 compares the detection of the 25 most relevant lines in Fig. 1 (a). Fig. 9 (left) shows the obtained result with the use of convolution, whereas for Fig. 9 (right) no convolution has been applied. Note the recovered line corresponding to the limit of the checkerboard on the top of Fig. 9 (left). Two segments define such a line, each one visible at one side of the dark king (Fig. 8, right). As the two segments fall on neighbor cells in the discretized parameter space (\(i.e., \text{voting map}\)), the convolution combines the two peaks
Algorithm 4 Function \( \text{Vote}() \). It complements the proposed voting process (Algorithm 3)

\begin{algorithm}[h]
\begin{algorithmic}[1]
\Require \text{Votes}, \( \rho_{\text{start}} \), \( \theta_{\text{start}} \), \( \delta_\rho \), \( \delta_\theta \), \( g_s \), \( S'_k \) \{Parameters\}
\State \( \rho_k, \theta_k, \sigma^2_{\rho}, \sigma_{\theta}, \sigma_{\rho_k \theta_k} \leftarrow \) Parameters computer for \( S'_k \)
\State \( R \leftarrow \sqrt{w^2 + h^2}/2 \)
\For \{Loop for the \( \theta \) coordinates of the parameter space\}
\State \( \theta_j \leftarrow \theta_{\text{start}} \)
\Repeat
\If \((\theta_j < 0^\circ) \text{ or } (\theta_j \geq 180^\circ)\)
\State \( \delta_\rho \leftarrow -\delta_\rho \)
\State \( \rho_k \leftarrow -\rho_k \)
\State \( \sigma_{\rho_k \theta_k} \leftarrow -\sigma_{\rho_k \theta_k} \)
\State \( \rho_{\text{start}} \leftarrow -\rho_{\text{start}} \)
\State \If \( \theta_j < 0^\circ \)
\State \( \theta_k \leftarrow 180^\circ - \delta_\theta + \theta_k \)
\State \( \theta_j \leftarrow 180^\circ - \delta_\theta \)
\Else
\State \( \theta_k \leftarrow \theta_k - 180^\circ + \delta_\theta \)
\State \( \theta_j \leftarrow 0^\circ \)
\EndIf
\EndIf
\For \{Loop for the \( \rho \) coordinates of the parameter space\}
\State \( g \leftarrow \text{round}(g_s G_k(\rho_j, \theta_j)) \) \{\( G_k \) from Eq. (15)\}
\While \((g > 0) \text{ and } (-R \leq \rho_j \leq R)\)
\State \( \text{Votes}(\rho_j, \theta_j) \leftarrow \text{Votes}(\rho_j, \theta_j) + g \)
\State \( \rho_j \leftarrow \rho_j + \delta_\rho \)
\State \( g \leftarrow \text{round}(g_s G_k(\rho_j, \theta_j)) \)
\EndWhile
\State \( \theta_j \leftarrow \theta_j + \delta_\theta \)
\Until no votes have been included by the internal loop
\end{algorithmic}
\end{algorithm}

into a larger one, resulting in a significant line. For the result shown in Fig. 9 (right), since no convolution has been applied, the two smaller peaks were both missed and replaced by the edges on the wall and on the chess horse, which are less relevant. The convolution operation is also applicable to the conventional voting scheme and was used to generate the results used for comparison with the proposed approach.

After the sorting step, we use a sweep plane that visits each cell of the list. By treating the parameter space as a height-field image, the sweeping plane gradually moves from each peak toward the zero height. For each visited cell, we check if any of its 8 neighbors has already been visited. If so, the current cell should be a smaller peak next to a taller one. In such a case, we mark the
current cell as visited and move the sweeping plane to the next cell in the list. Otherwise, we add the current cell to the list of detected peaks, mark it as visited and then proceed with the sweeping plane scan to the next cell in the list. The resulting group of detected peaks contains the most significant lines identified in the image, already sorted by number of votes.

4 Results

In order to compare the performances of the gradient-based (i.e., the universally used version of the Hough transform) and the proposed Hough transform voting schemes, both techniques were implemented using C++ and tested over a set of images of various resolutions, some of which are shown in Figs. 1, 10 and 11. Edge extraction and thinning were performed with standard image processing techniques using Matlab 7.0. The resulting feature pixels were used as input for the gradient-based Hough transform (GHT) algorithm, which has been optimized with the use of lookup tables. It was also assumed that each edge pixel votes for a range of 20 angular values, computed from its gradient information. The C++ code was compiled using Visual Studio 2003 as dynamic link libraries (DLLs) so that they could be called from Matlab. The C++ code for the proposed approach (KHT) also handles the linking of adjacent edge-pixel chains (Appendix 1) and their subdivision into the most perceptually-significant straight line segments [27]. After the voting process, the peaks in the parameter space are identified using the technique described in Section 3.4. The reported experiments were performed on an AMD Athlon 64 3700+ (2.21 GHz) with 2 GB of RAM. However, due to some constraints imposed by the available Matlab version, the code was compiled for 32 bits.
Fig. 9. The 25 most relevant detected lines for the image shown in Fig. 1 (a) obtained using our kernel-based voting and peak detection algorithms. On the left, a 3x3 Gaussian kernel was convolved with the voting map before performing peak detection. On the right, no convolution was applied prior to peak detection. Not being able to exploit the full potential of the hardware.

Tables 1 and 2 summarize the results obtained on seven groups of images (Figs. 1, 10 and 11) resampled at various resolutions. For each image, the intermediate-resolution version was obtained by cropping the full-resolution one, and the lower resolution was obtained by resampling the cropped version. The complexity of these images are indicated by their number and distribution of the feature pixels, which are depicted in Fig. 1 (b) and in the second row of both Figs. 10 and 11, respectively. The images shown in Fig. 10 (left) and Fig. 11 were chosen because they represent challenging examples. For these, the results produced by the conventional voting scheme (Fig. 1 (c) and third row of both Figs. 10 and 11) lend to many variants (alias) of the same lines, leaving many other important ones unreported. This situation becomes evident on the right side of the Road where multiple concurrent lines were extracted, leaving important lines undetected on the painted pavement. The same problem is observed on the bottom of the Church and Building images in Fig. 11. Note that Figs. 10 (center) and (right) contain much cleaner sets of straight line segments, which tends to simplify the line detection task. These images were used to demonstrate that the conventional brute-force voting approach tends to create spurious peaks and identify several variations of the same line, even when applied to simple configurations. As a result, it failed to report some important lines as significant ones. For instance, note the undetected diagonal lines on the four corners of the Wall image (Fig. 10). In comparison, the proposed approach (Fig. 1 (d) and fourth row of both Figs. 10 and 11) was capable of identifying all important lines in all seven images. Since applications that use Hough transform usually have to specify the number of lines to be extracted, robustness against the detection of false positives is of paramount importance.

In Table 1, the third column (Edge Pixels) shows the number of extracted
Fig. 10. Comparison of the line-detection results produced by the gradient-based (GHT) and the proposed (KHT) Hough transform voting and line detection schemes. Set of test images (top). From left to right, these images correspond to the Road, Wall and Board datasets referenced in Tables 1 and 2 and exhibit the 15, 36 and 38 most important lines detected by each algorithm, respectively. Note that in all examples the GHT did not detect important lines, which were detected by the KHT (bottom).

feature pixels in the images. The fourth and fifth columns (Edge Groups) show the numbers of identified clusters and the numbers of clusters that passed the
Fig. 11. Comparison of the line-detection results produced by the gradient-based (GHT) and the proposed (KHT) Hough transform voting and line detection schemes. The first row shows the original images, whose feature pixels, which were detected using a Canny edge detector, are shown on the second row. From left to right, these images correspond to the Church, Building and Beach datasets referenced in Tables 1 and 2 and exhibit the 40, 19 and 19 most important lines detected by each algorithm, respectively. Rows three and four show the most relevant lines detected by GHT and KHT, respectively. Note the important lines undetected by the GHT.
culling optimization described in Section 3.3. Columns six and seven of Table 1 (Used Pixels) show the actual numbers of feature pixels processed by the KHT approach and their percentage with respect to the entire image. In Table 2, the ‘GHT Voting’ and ‘KHT voting’ columns show the time (in milliseconds) involved in the execution of the GHT and KHT, respectively. One should note that the time for KHT includes the processing required to link and subdivide clusters of neighbor pixels, which is not necessary for the GHT. The last four columns of Table 2 show the number of detected peaks by both approaches and the corresponding detection times. These times were computed by averaging the results of 50 executions of the reported tasks. A detailed account for the times involved in each step of the proposed voting strategy are presented in Table 3 for various resolutions of the images in Fig. 1, 10 and 11.

Table 2 shows that the proposed approach can achieve up to 200 fps for images with $512 \times 512$ (e.g., Board and Building) without performing peak detection, and up to 125 fps with peak detection. For the same Board and Building images, respectively, the gradient-based Hough transform executes at 58.82 fps and 45.45 fps without peak detection, and 10.75 fps and 9.71 fps, with peak detection. When considering only the computation of the Gaussian kernel parameters and voting, KHT can process images with $1600 \times 1200$ pixels (e.g., Wall) at 100 fps, and with $512 \times 512$ pixels at 200 fps. Including the time required to link and subdivide cluster of pixels (but without peak detection), KHT can still process a $1600 \times 1200$ image at 34.48 fps. These results are superior to the ones reported by [17] for 31 of the 34 analyzed architectures for real-time Hough transform, and comparable to the other three.

Fig. 12 provides a graphical comparison between the performances of the gradient-based Hough transform and our approach. The horizontal axis enumerates the set of input images of various resolutions and the vertical axis shows the execution times in milliseconds. Note that KHT is always much faster than GHT (KHT’s entire process takes less than GHT’s voting time) and processes almost all images under the mark of 33 ms (30 Hz, represented by the dotted line). The only exceptions were high resolution versions of Road, Wall and Beach, which took 34, 36 and 36 ms, respectively.

A careful inspection of Table 2 reveals that although the total time (i.e., voting time plus peak detection time) for KHT correlates with the size of the images, the peak detection time for the $512 \times 512$ version of the Chess image took longer (extra 4 ms) than for the higher resolution versions of the same image. This can be explained considering that many small groups of non-collinear feature pixels are found in the Chess image (Fig. 8, left). As the image was filtered down, the relative uncertainty in the resulting feature pixels increased due to the filtering itself and to the decrease in resolution. As a result, the peak of the highest Gaussian used for normalizing the heights of all kernels became smaller, thus allowing a larger number of low-height kernels to pass
<table>
<thead>
<tr>
<th>Image</th>
<th>Resolution</th>
<th>Edge Pixels</th>
<th>Edge Groups</th>
<th>Used Pixels</th>
<th>Count</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chess</td>
<td>1280 × 960</td>
<td>70,238</td>
<td>2,756</td>
<td>110</td>
<td>12,717</td>
<td>1.03%</td>
</tr>
<tr>
<td></td>
<td>960 × 960</td>
<td>52,547</td>
<td>2,035</td>
<td>172</td>
<td>12,136</td>
<td>1.32%</td>
</tr>
<tr>
<td></td>
<td>512 × 512</td>
<td>18,653</td>
<td>765</td>
<td>216</td>
<td>8,704</td>
<td>3.32%</td>
</tr>
<tr>
<td>Road</td>
<td>1280 × 960</td>
<td>101,756</td>
<td>4,069</td>
<td>170</td>
<td>16,099</td>
<td>1.31%</td>
</tr>
<tr>
<td></td>
<td>960 × 960</td>
<td>75,281</td>
<td>2,981</td>
<td>151</td>
<td>13,814</td>
<td>1.50%</td>
</tr>
<tr>
<td></td>
<td>512 × 512</td>
<td>19,460</td>
<td>698</td>
<td>127</td>
<td>7,677</td>
<td>2.93%</td>
</tr>
<tr>
<td>Wall</td>
<td>1600 × 1200</td>
<td>72,517</td>
<td>2,050</td>
<td>414</td>
<td>36,679</td>
<td>1.91%</td>
</tr>
<tr>
<td></td>
<td>960 × 960</td>
<td>38,687</td>
<td>1,026</td>
<td>267</td>
<td>22,604</td>
<td>2.45%</td>
</tr>
<tr>
<td></td>
<td>512 × 512</td>
<td>16,076</td>
<td>438</td>
<td>230</td>
<td>11,870</td>
<td>4.53%</td>
</tr>
<tr>
<td>Board</td>
<td>1280 × 960</td>
<td>66,747</td>
<td>2,292</td>
<td>162</td>
<td>19,023</td>
<td>1.55%</td>
</tr>
<tr>
<td></td>
<td>960 × 960</td>
<td>44,810</td>
<td>1,467</td>
<td>142</td>
<td>16,150</td>
<td>1.75%</td>
</tr>
<tr>
<td></td>
<td>512 × 512</td>
<td>15,775</td>
<td>453</td>
<td>99</td>
<td>7,435</td>
<td>2.84%</td>
</tr>
<tr>
<td>Church</td>
<td>768 × 1024</td>
<td>67,140</td>
<td>2,532</td>
<td>305</td>
<td>20,472</td>
<td>2.60%</td>
</tr>
<tr>
<td></td>
<td>768 × 768</td>
<td>45,372</td>
<td>1,529</td>
<td>242</td>
<td>17,813</td>
<td>3.02%</td>
</tr>
<tr>
<td></td>
<td>512 × 512</td>
<td>18,934</td>
<td>644</td>
<td>166</td>
<td>9,811</td>
<td>3.74%</td>
</tr>
<tr>
<td>Building</td>
<td>1024 × 768</td>
<td>64,453</td>
<td>2,379</td>
<td>57</td>
<td>10,826</td>
<td>1.38%</td>
</tr>
<tr>
<td></td>
<td>768 × 768</td>
<td>51,181</td>
<td>1,917</td>
<td>57</td>
<td>9,792</td>
<td>1.66%</td>
</tr>
<tr>
<td></td>
<td>512 × 512</td>
<td>21,568</td>
<td>748</td>
<td>63</td>
<td>6,729</td>
<td>2.57%</td>
</tr>
<tr>
<td>Beach</td>
<td>869 × 1167</td>
<td>105,589</td>
<td>4,564</td>
<td>210</td>
<td>14,785</td>
<td>1.46%</td>
</tr>
<tr>
<td></td>
<td>800 × 800</td>
<td>63,164</td>
<td>2,629</td>
<td>150</td>
<td>10,904</td>
<td>1.70%</td>
</tr>
<tr>
<td></td>
<td>512 × 512</td>
<td>24,880</td>
<td>1,019</td>
<td>42</td>
<td>3,873</td>
<td>1.48%</td>
</tr>
</tbody>
</table>

Note that the number of detected peaks (column 7 in Table 2) is bigger for the lower-resolution version of the Chess image than for its higher resolution versions. A similar phenomenon can be observed for other images listed in the same table, but at reduced scale.

For GHT the number of accesses to the voting map grows with the number of feature points, the image dimensions, the discretization step $\delta$, and the range $r$ of selected $\theta$ values over which the associated $\rho$ values should be calculated. Such number of access can be expressed as $q' = rn/\delta$, where $n$ is the total
Table 2
Comparison of the performance of the gradient-based (GHT) and proposed (KHT) algorithms for the steps of voting and peak detection on the images shown in Figs. 1, 10 and 11. For KHT, voting time includes clustering approximately collinear feature pixels into segments. Time is expressed in milliseconds (ms).

<table>
<thead>
<tr>
<th>Image</th>
<th>Resolution</th>
<th>GHT Voting</th>
<th>KHT Voting</th>
<th>GHT Peaks</th>
<th>KHT Peaks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>ms</td>
<td>ms</td>
<td>Count</td>
<td>ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chess</td>
<td>1280 × 960</td>
<td>83</td>
<td>21</td>
<td>42,060</td>
<td>223</td>
</tr>
<tr>
<td></td>
<td>960 × 960</td>
<td>60</td>
<td>17</td>
<td>35,779</td>
<td>179</td>
</tr>
<tr>
<td></td>
<td>512 × 512</td>
<td>20</td>
<td>9</td>
<td>18,992</td>
<td>82</td>
</tr>
<tr>
<td>Road</td>
<td>1280 × 960</td>
<td>123</td>
<td>27</td>
<td>47,414</td>
<td>257</td>
</tr>
<tr>
<td></td>
<td>960 × 960</td>
<td>87</td>
<td>20</td>
<td>40,513</td>
<td>217</td>
</tr>
<tr>
<td></td>
<td>512 × 512</td>
<td>20</td>
<td>6</td>
<td>22,814</td>
<td>90</td>
</tr>
<tr>
<td>Wall</td>
<td>1600 × 1200</td>
<td>81</td>
<td>29</td>
<td>48,648</td>
<td>238</td>
</tr>
<tr>
<td></td>
<td>960 × 960</td>
<td>42</td>
<td>14</td>
<td>34,570</td>
<td>144</td>
</tr>
<tr>
<td></td>
<td>512 × 512</td>
<td>17</td>
<td>8</td>
<td>17,213</td>
<td>65</td>
</tr>
<tr>
<td>Board</td>
<td>1280 × 960</td>
<td>78</td>
<td>21</td>
<td>44,438</td>
<td>223</td>
</tr>
<tr>
<td></td>
<td>960 × 960</td>
<td>51</td>
<td>15</td>
<td>39,280</td>
<td>173</td>
</tr>
<tr>
<td></td>
<td>512 × 512</td>
<td>17</td>
<td>5</td>
<td>21,605</td>
<td>76</td>
</tr>
<tr>
<td>Church</td>
<td>768 × 1024</td>
<td>73</td>
<td>21</td>
<td>33,088</td>
<td>175</td>
</tr>
<tr>
<td></td>
<td>768 × 768</td>
<td>49</td>
<td>14</td>
<td>28,714</td>
<td>138</td>
</tr>
<tr>
<td></td>
<td>512 × 512</td>
<td>20</td>
<td>7</td>
<td>17,843</td>
<td>77</td>
</tr>
<tr>
<td>Building</td>
<td>1024 × 768</td>
<td>70</td>
<td>17</td>
<td>35,736</td>
<td>181</td>
</tr>
<tr>
<td></td>
<td>768 × 768</td>
<td>54</td>
<td>13</td>
<td>29,720</td>
<td>146</td>
</tr>
<tr>
<td></td>
<td>512 × 512</td>
<td>22</td>
<td>5</td>
<td>18,510</td>
<td>81</td>
</tr>
<tr>
<td>Beach</td>
<td>869 × 1167</td>
<td>120</td>
<td>28</td>
<td>37,879</td>
<td>222</td>
</tr>
<tr>
<td></td>
<td>800 × 800</td>
<td>70</td>
<td>17</td>
<td>30,976</td>
<td>166</td>
</tr>
<tr>
<td></td>
<td>512 × 512</td>
<td>26</td>
<td>6</td>
<td>19,276</td>
<td>92</td>
</tr>
</tbody>
</table>

number of pixels in the image and $0^\circ \leq r < 180^\circ$. For KHT, the number of accesses depends on the number $m$ of clusters that cast votes and the areas of their kernel footprints. Given an average footprint area $s$ (expressed in cells of the voting map), the number of accesses is given by $q'' = sm/\delta$. Thus, KHT has a smaller number of accesses whenever $sm < rn$, which tends to always happen, since $s \ll n$.  

24
Table 3
Processing time for all steps of KHT’s voting procedure.

<table>
<thead>
<tr>
<th>Image</th>
<th>Resolution</th>
<th>Link</th>
<th>Subdivide</th>
<th>Kernel</th>
<th>Vote</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chess</td>
<td>1280 × 960</td>
<td>9</td>
<td>7</td>
<td>5</td>
<td>&lt; 1</td>
</tr>
<tr>
<td></td>
<td>960 × 960</td>
<td>7</td>
<td>5</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>512 × 512</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Road</td>
<td>1280 × 960</td>
<td>11</td>
<td>7</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>960 × 960</td>
<td>8</td>
<td>5</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>512 × 512</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Wall</td>
<td>1600 × 1200</td>
<td>12</td>
<td>7</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>960 × 960</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>512 × 512</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Board</td>
<td>1280 × 960</td>
<td>9</td>
<td>7</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>960 × 960</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>512 × 512</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>Church</td>
<td>768 × 1024</td>
<td>7</td>
<td>5</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>768 × 768</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>512 × 512</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Building</td>
<td>1024 × 768</td>
<td>7</td>
<td>5</td>
<td>5</td>
<td>&lt; 1</td>
</tr>
<tr>
<td></td>
<td>768 × 768</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>&lt; 1</td>
</tr>
<tr>
<td></td>
<td>512 × 512</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>Beach</td>
<td>869 × 1167</td>
<td>11</td>
<td>7</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>800 × 800</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>512 × 512</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>&lt; 1</td>
</tr>
</tbody>
</table>

Although no pre-processing was applied to the images used in our experiments, we noticed that the presence of severe radial and tangential distortions [2] may affect the quality of the results produced by KHT. This can be explained as the fact that straight lines might become curved toward the borders of the image, resulting in possibly large sets of small line segments, which will tend to spread their votes across the parameter space. This problem also affects other variants of the Hough transform, and can be easily avoided by pre-warping the images to remove these distortions. Such a warping can be effectively implemented for any given lens-camera setup with the use of lookup tables.
Fig. 12. Comparison between the processing times of the gradient-based Hough transform (GHT) and of the proposed approach (KHT). For each input image (horizontal axis), the left bar represents GHT and the right one represents KHT. The height of the bar is given by the sum of the time spent by each step of the corresponding approach. The dotted line defines the mark of 33 ms, approximately 30 Hz.

A limitation of the proposed technique is the lack of detection of thin dotted lines, where the grouping algorithm is not capable of identifying strings of neighboring pixels. Although in practice we have not experienced this kind of problem, it may happen and can be avoided by changing the strategy adopted for linking such pixels, or by filtering the image down before applying the algorithm. In our current implementation of the described algorithm, we have chosen to group approximately collinear feature pixels using the linking procedure from [26] and the subdivision scheme from [27] because they are fast and handle the general case. However, other approaches for chaining and subdivision can be used for achieving specific results, such as handling the dotted line example just mentioned.

4.1 Memory Utilization

The amount of memory required by the GHT consists of the voting map and lookup tables used to optimize the transform implementation. In the case of the KHT, no lookup tables are necessary, but it requires 6 floats for each elliptical-Gaussian kernel: \( \rho \) and \( \theta \) (the ‘center’ of the ellipse), \( \sigma_{\rho}^2 \) and \( \sigma_{\theta}^2 \) and
σρθ, and the height of the kernel (used to optimize the culling of the small kernels). Thus, assuming an image with 1600 × 1200 pixels and δ = 0.5, the voting map for both approaches would require 10.99 MB. The lookup tables for the GHT would take 2.81 KB, whereas the extra information required by the KHT would need 23.44 KB. This minor increase on memory needs of the KHT is more than compensated by the significant speedups obtained by the method.

5 Conclusions and Future Work

We have presented an improved voting scheme for the Hough transform that allows a software implementation to achieve real-time performance even for relatively large images. The approach clusters approximately collinear edge pixels and, for each cluster, casts votes for a reduced set of lines in the parameter space, based on the quality of the fitting of a straight line to the pixels of the cluster. The voting process is weighted using oriented elliptical-Gaussian kernels, significantly accelerating the process and producing a much cleaner voting map. As a result, our approach is extremely robust to the detection of spurious lines, very common in the traditional approach.

The KHT is not only significantly faster than previous known software implementations of the Hough transform, but its performance also compares favorably to the reported results of most specialized hardware architectures for HT.

We also presented a simple sweep-plane-based approach for identify peaks of votes in the parameter space. We believe that these ideas may lend to optimizations on several computer vision, image processing and computer graphics applications that require real-time detection of lines. We are exploring ways to extend our kernel-based voting scheme to detect arbitrary curves and objects of known geometry.

Acknowledgements

This work was partially sponsored by CNPq - Brazil (477344/2003-8) and Petrobras (502009/2003-9). The authors would like to thank Microsoft Brazil for additional support, Prof. Roberto da Silva (PPGC-UFRGS) for some discussions, and the anonymous reviewers for their comments and insightful suggestions.
References


Appendix 1: Linking procedure

The linking procedure (Algorithm 5) creates a string of neighboring edge pixels. It takes as input a binary image and a reference feature pixel from which one would like to retrieve a pixel chain. Note that this algorithm destroys the original binary image (lines 6 and 14). The pseudo code shown in Algorithm 5 was inspired on an implementation available in the VISTA image-processing library [26].

Algorithm 5 Linking of neighboring edge pixels into strings. The pseudo code for the function Next () is presented in Algorithm 6.

<table>
<thead>
<tr>
<th>Require: I {Binary image}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Require: p_{\text{ref}} {A feature pixel}</td>
</tr>
<tr>
<td>1: S ← ∅ {Create an empty string of pixels}</td>
</tr>
<tr>
<td>2: {Find and add feature pixels to the end of the string}</td>
</tr>
<tr>
<td>3: p ← p_{\text{ref}}</td>
</tr>
<tr>
<td>4: repeat</td>
</tr>
<tr>
<td>5: S ← S + p</td>
</tr>
<tr>
<td>6: I(x_p, y_p) ← 0</td>
</tr>
<tr>
<td>7: p ← Next (p, I)</td>
</tr>
<tr>
<td>8: until p be an invalid pixel position</td>
</tr>
<tr>
<td>9: {Find and add feature pixels to the begin of the string}</td>
</tr>
<tr>
<td>10: p ← Next (p_{\text{ref}}, I)</td>
</tr>
<tr>
<td>11: if p is a valid pixel position then</td>
</tr>
<tr>
<td>12: repeat</td>
</tr>
<tr>
<td>13: S ← p + S</td>
</tr>
<tr>
<td>14: I(x_p, y_p) ← 0</td>
</tr>
<tr>
<td>15: p ← Next (p, I)</td>
</tr>
<tr>
<td>16: until p be an invalid pixel position</td>
</tr>
<tr>
<td>17: end if</td>
</tr>
</tbody>
</table>

Algorithm 6 Function Next (). It complements the linking procedure (Algorithm 5).

<table>
<thead>
<tr>
<th>Require: I {Binary image}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Require: p_{\text{seed}} {Current feature pixel}</td>
</tr>
<tr>
<td>1: for each neighbor pixel of p_{\text{seed}} do</td>
</tr>
<tr>
<td>2: p ← current neighbor pixel of p_{\text{seed}}</td>
</tr>
<tr>
<td>3: if p is a feature pixel then</td>
</tr>
<tr>
<td>4: Break the loop and return p {The next pixel was found}</td>
</tr>
<tr>
<td>5: end if</td>
</tr>
<tr>
<td>6: end for</td>
</tr>
<tr>
<td>7: Return an invalid pixel position {The next pixel was not found}</td>
</tr>
</tbody>
</table>